State Notation Language and the Sequencer

NSLS-II EPICS Training

Ralph Lange <rlange@bnl.gov>

Acknowledgements

- Slides for this presentation have been taken from talks prepared by the following people:
    - Andrew Johnson (Argonne)
    - Bob Dalesio (LANL/SNS/LCLS/BNL)
    - Deb Kerstiens (LANL)
    - Rozelle Wright (LANL)
    - Ned Arnold (Argonne)
    - John Maclean (Argonne)

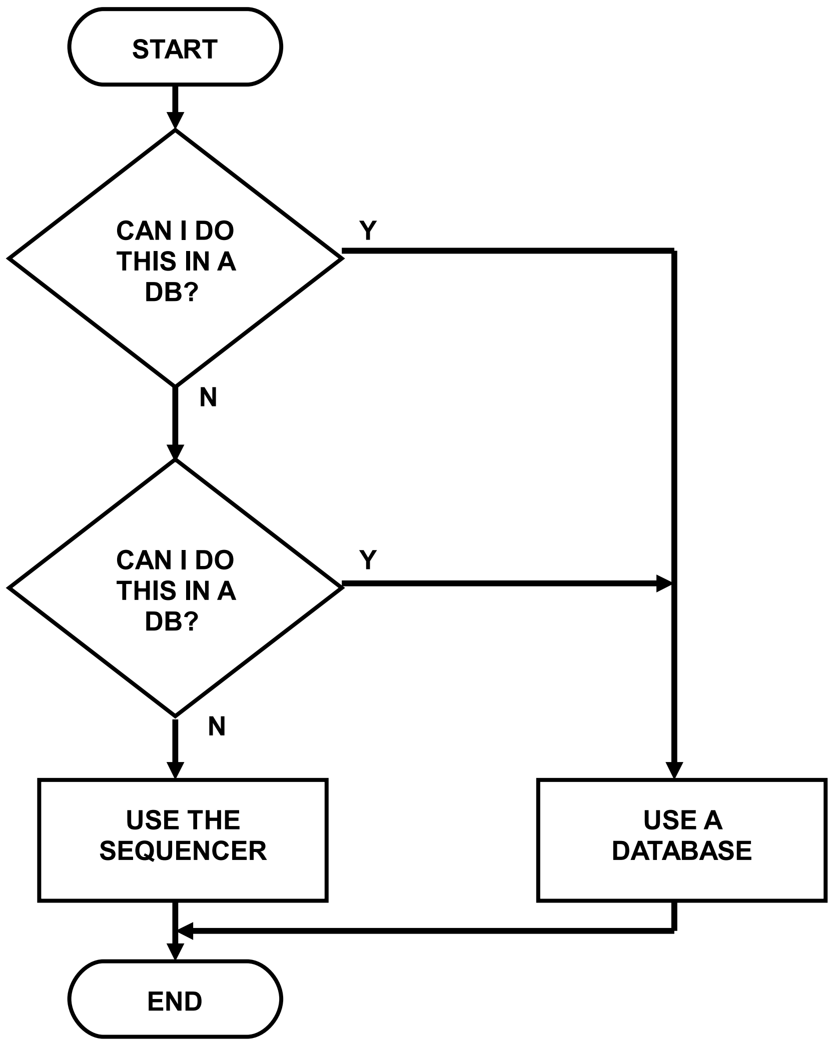
SNL and the Sequencer

- The sequencer runs programs written in State Notation Language (SNL)
- SNL is a ‘C’ like language to facilitate programming of sequential operations
- Fast execution - compiled code
- Programming interface to extend EPICS in the real-time environment
- Common uses
  - Provide automated start-up sequences like vacuum or RF where subsystems need coordination
  - Provide fault recovery or transition to a safe state
  - Provide automatic calibration of equipment

Advantages of SNL

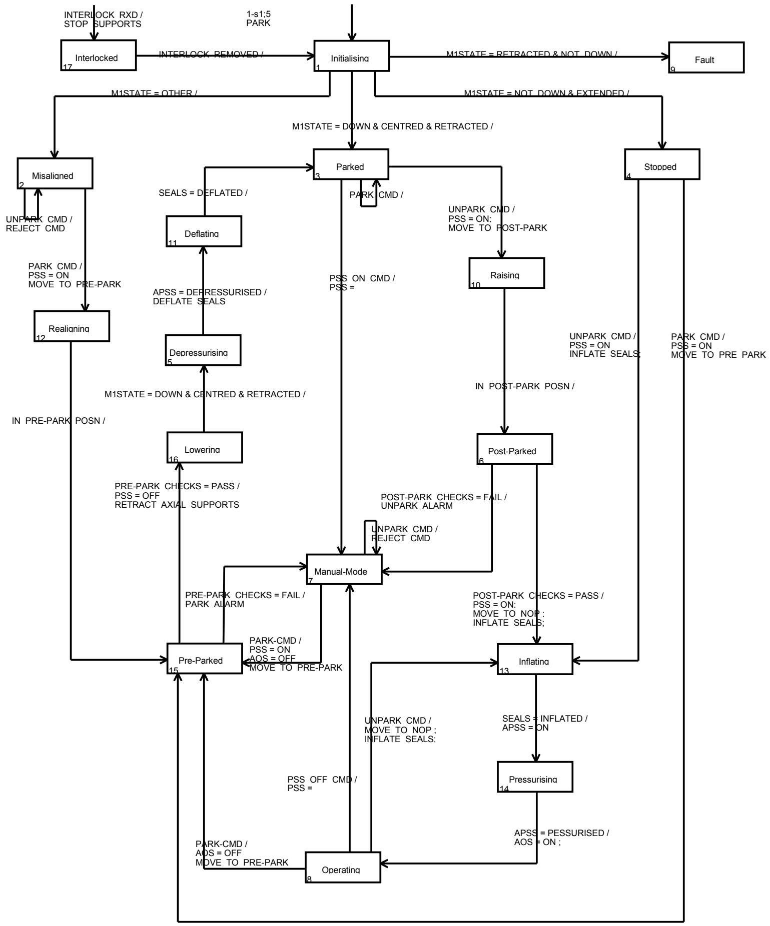
- Can implement complicated algorithms
- Can stop, reload, restart a sequence program without rebooting
- Interacts with the operator through string records and mbbo records
- C code can be embedded as part of the sequence
- All Channel Access details are taken care of for you
- File access can be implemented as part of the sequence

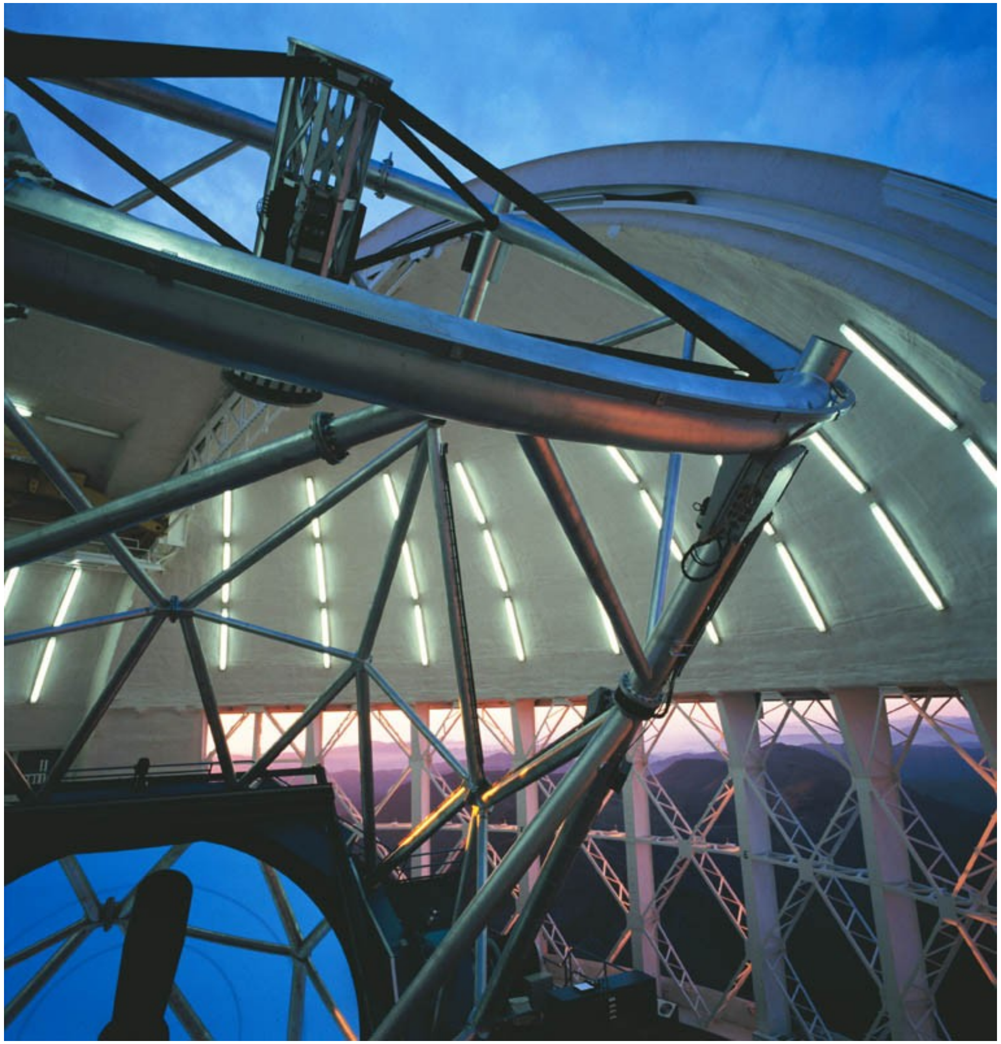
Should I Use the Sequencer?



When to Use the Sequencer

- For sequencing complex events, e.g. parking and unparking a telescope mirror

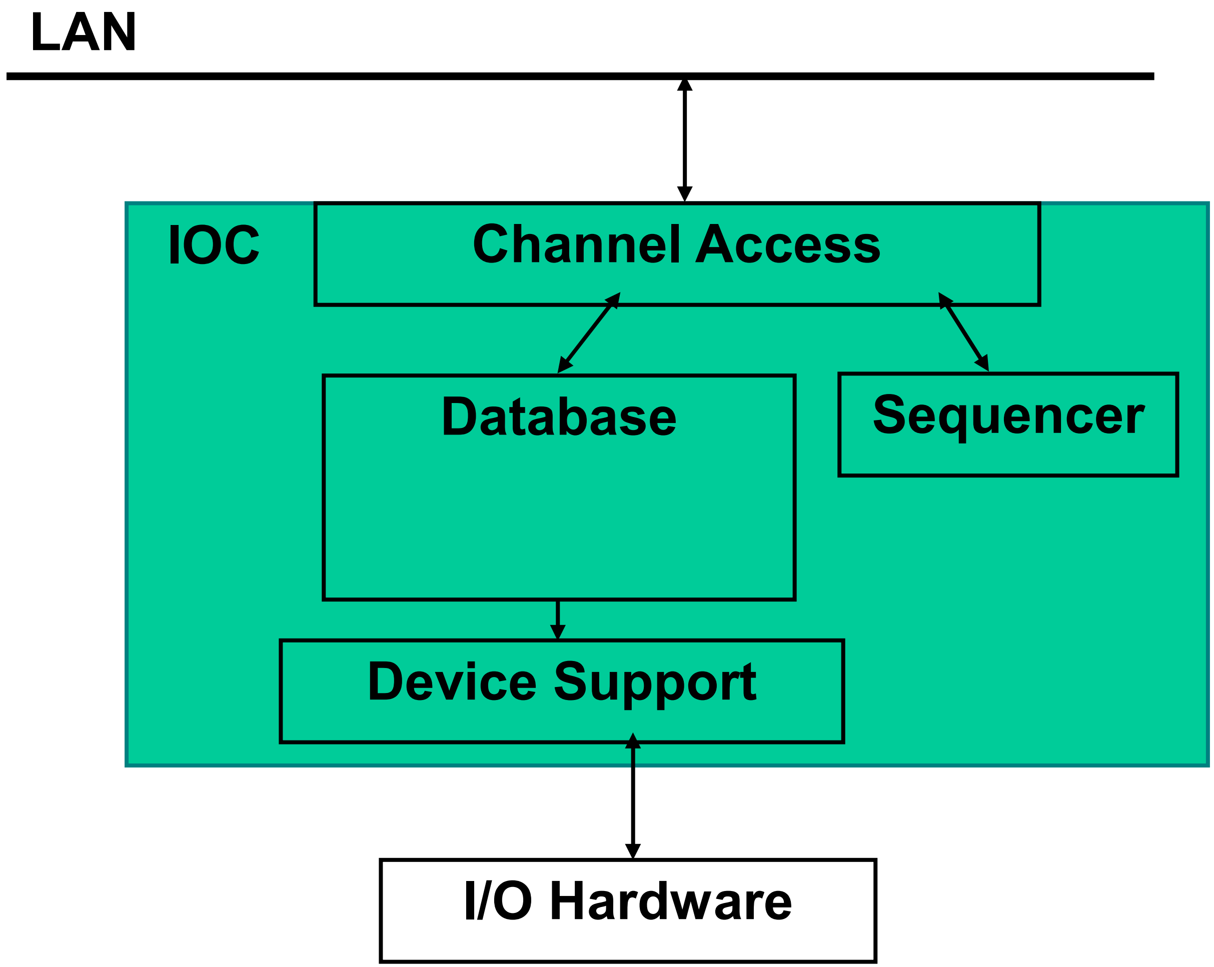




Photograph courtesy of the Gemini Telescopes project

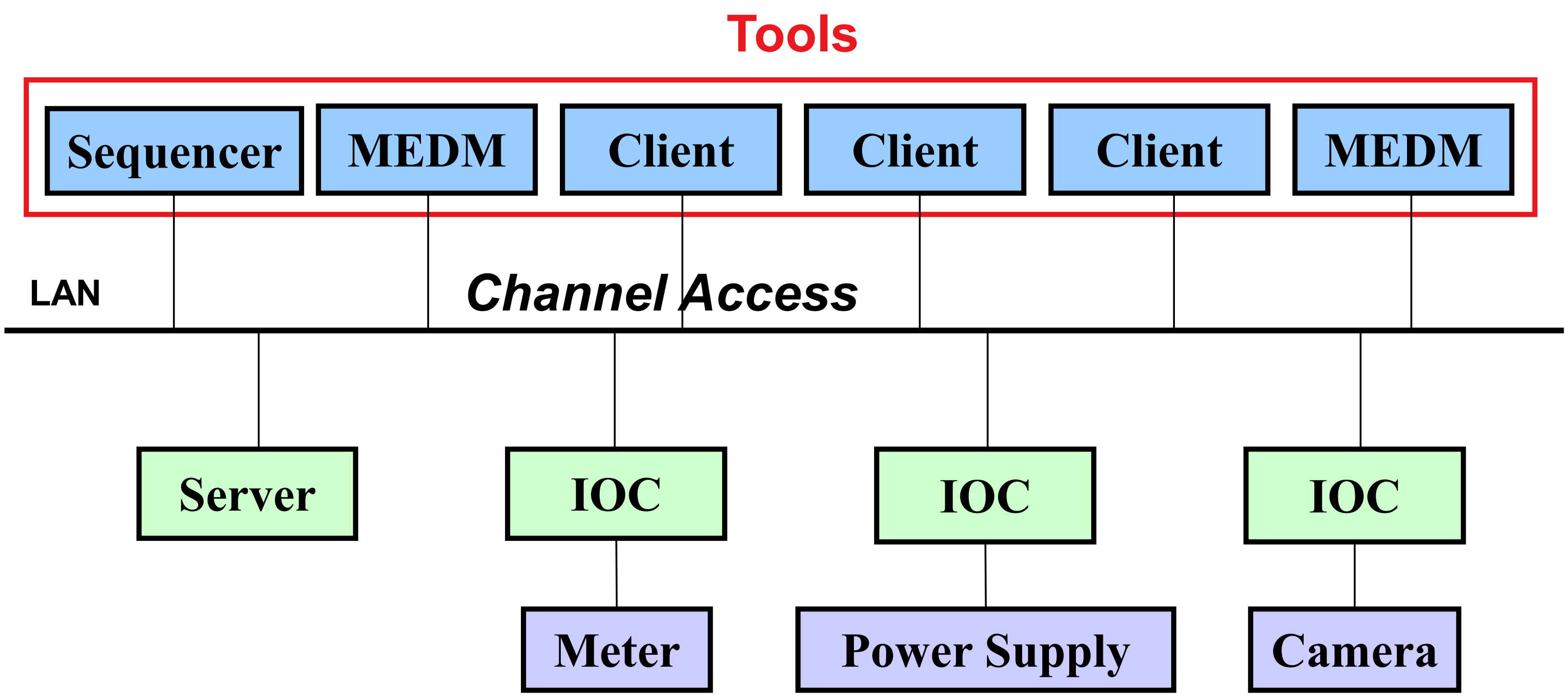
Where is the Sequencer?

- On the IOC:



Where is the Sequencer?

- On the workstation:



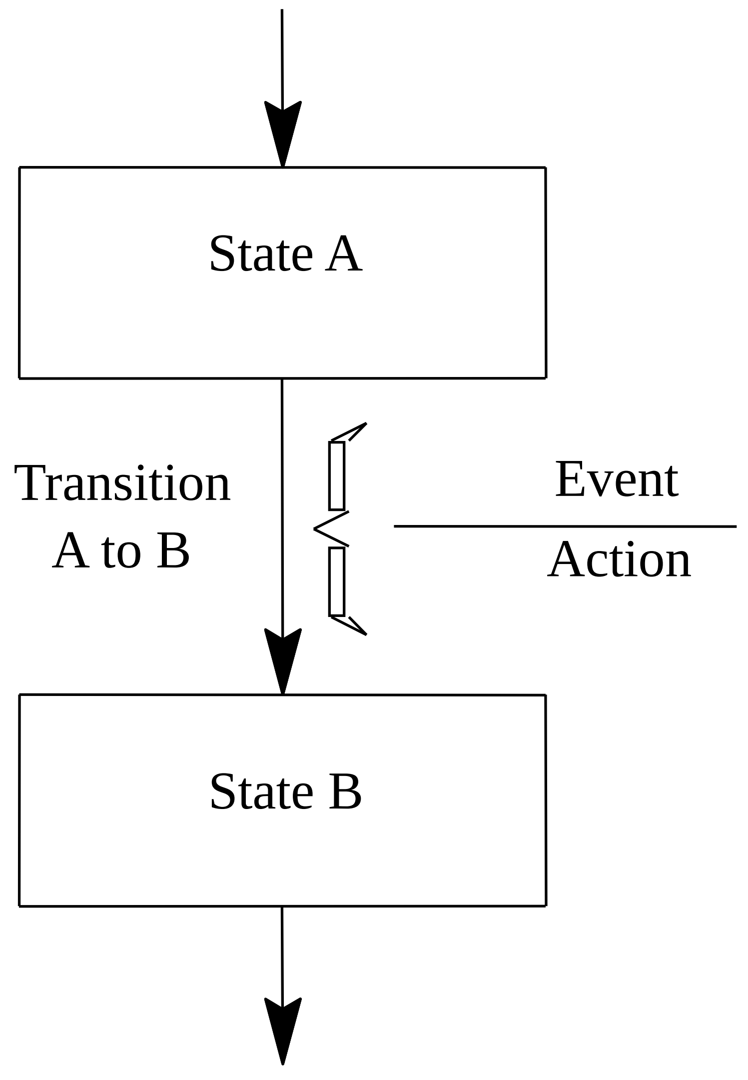
The Best Place for the Sequencer

- Traditionally, sequencers run in the IOC
  Recent versions of the sequencer can be run either in an IOC or as a standalone program on a workstation

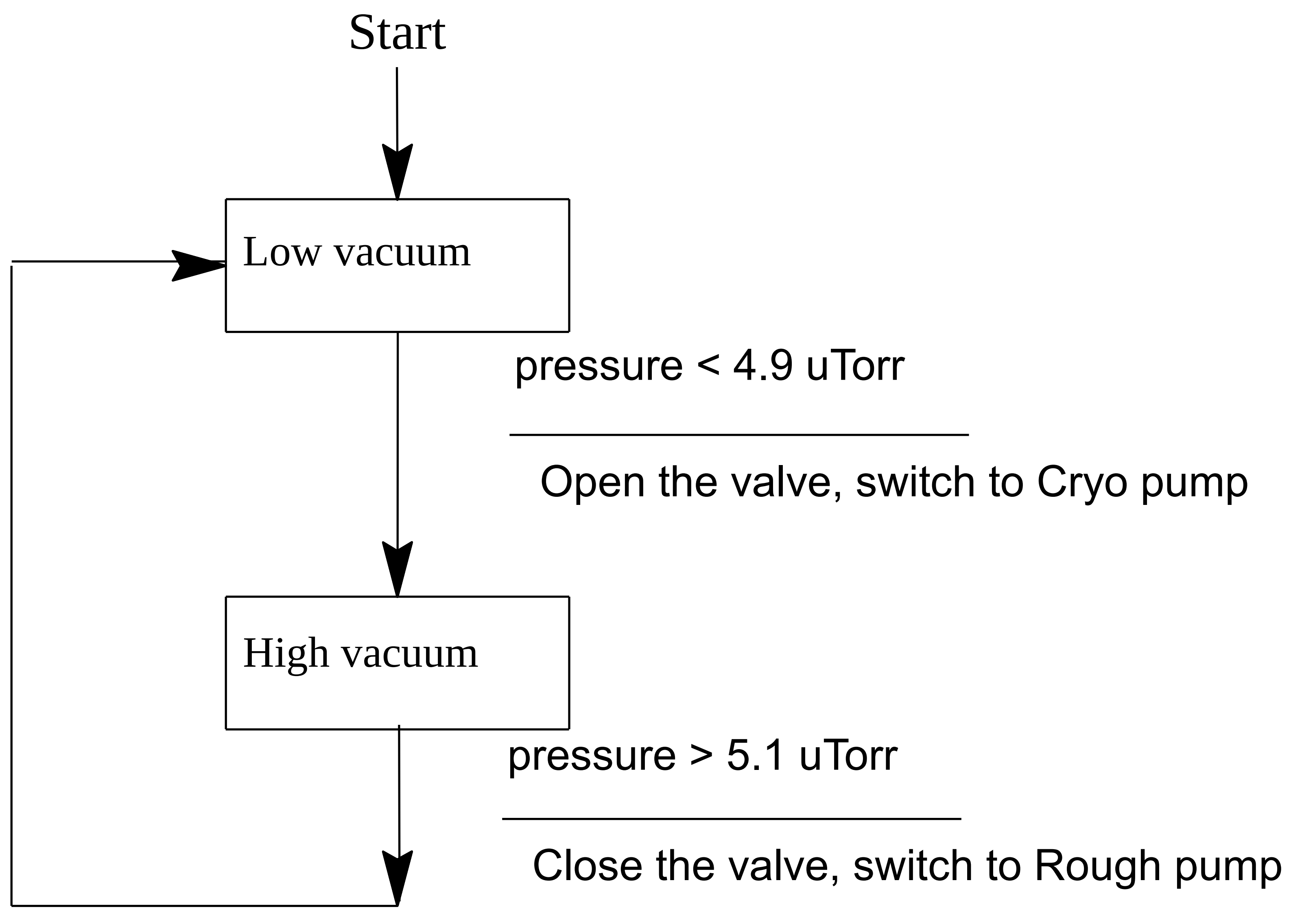
- Locating them within the IOC they control makes them easier to manage and independent from network issues
- Running them on a workstation can make testing and debugging easier

- On a workstation, SNL provides an easy way to write simple CA client programs

SNL Implements State Transition Diagrams



Example – State Transition Diagram



SNL – General Structure and Syntax

```
program program_name
declarations

ss state_set_name {

    state state_name {

        entry {
            entry action statements
        }

        when (event) {
            action statements
        } state next_state_name

        when (event) {
            ...
        } state next_state_name

        exit{
            exit action statements
        }
    }
    state state_name {
        ...
    }
}
```

A program may contain multiple state sets.

A state set becomes a task or thread.

A state is an area where the task waits for events. The first state defined in a state set is the initial state.

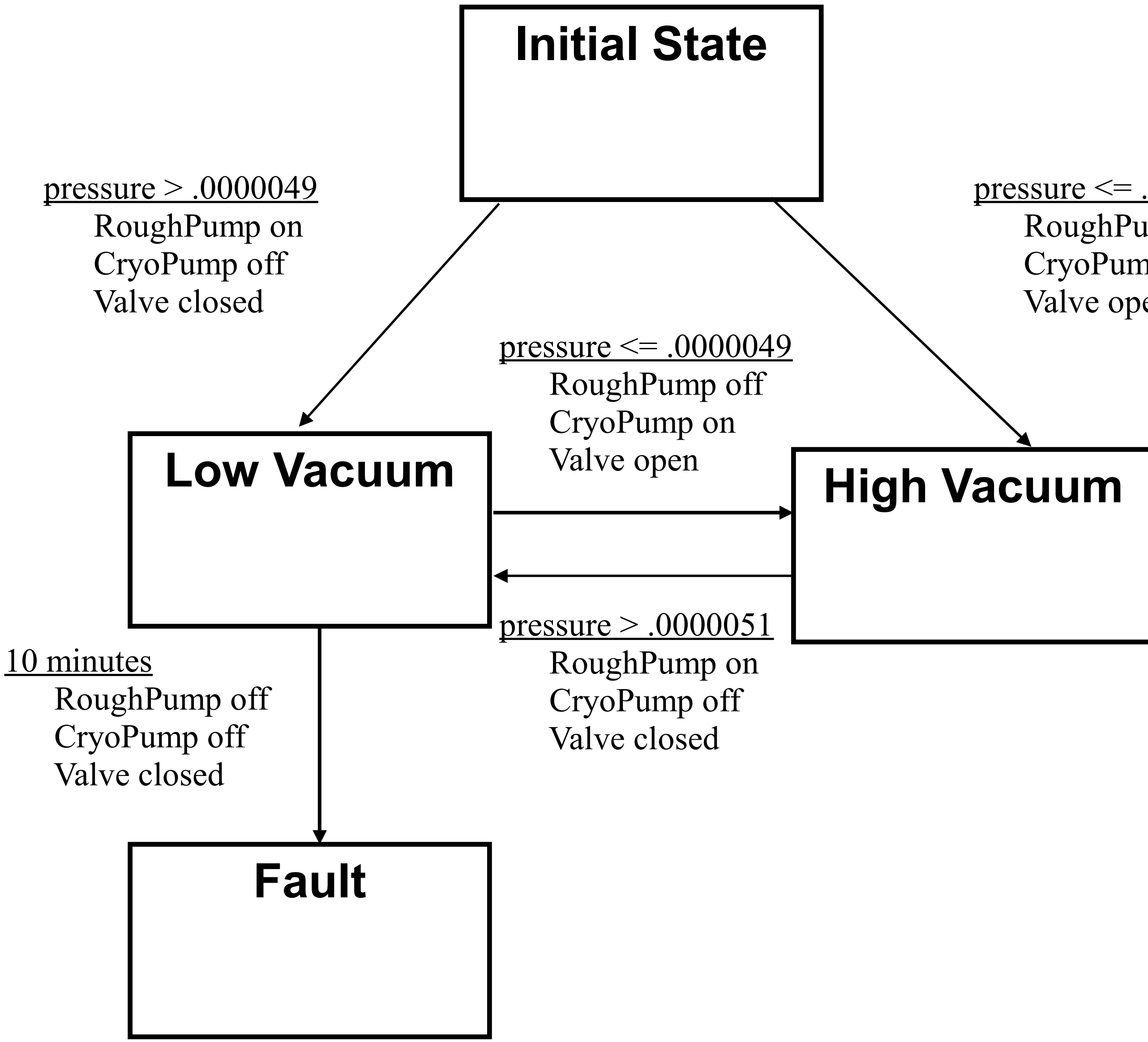
Actions to do on entry to this state from another state.

Defines an event for which this state waits and actions to do when the event occurs.

Specifies the following state after the actions complete.

Actions to do before exiting this state to another state.

Example – State Definitions and Transitions



Example - Declarations

```
double  pressure;
assign  pressure to "Tank1Coupler1PressureRB";
monitor pressure;

short   RoughPump;
assign  RoughPump to "Tank1Coupler1RoughPump";
short   CryoPump;
assign  CryoPump to "Tank1Coupler1CryoPump";
short   Valve;
assign  Valve to "Tank1Coupler1IsolationValve";
string  CurrentState;
assign  CurrentState to "Tank1Coupler1VacuumState";
```

Example – State Transitions (w/o Actions)

```
program vacuum_control

ss coupler_control
{
    state init {
        when (pressure > .0000049) {
        } state low_vacuum
        when (pressure <= .0000049) {
        } state high_vacuum
    }
    state high_vacuum {
        when (pressure > .0000051) {
        } state low_vacuum
    }
    state low_vacuum {
        when (pressure <= .0000049) {
        } state high_vacuum
        when (delay(600.0)) {
        } state fault
    }
    state fault {
    }
}
```

Example – Initial State

```
state init {
    entry {
        strcpy(CurrentState, ”Init”);
        pvPut(CurrentState);
    }

    when (pressure > .0000049) {
        RoughPump = 1;
        pvPut(RoughPump);
        CryoPump = 0;
        pvPut(CryoPump);
        Valve = 0;
        pvPut(Valve);
    } state low_vacuum

    when (pressure <= .0000049) {
        RoughPump = 0;
        pvPut(RoughPump);
        CryoPump = 1;
        pvPut(CryoPump);
        Valve = 1;
        pvPut(Valve);
    } state high_vacuum
}
```

Example – State low_vacuum

```
state low_vacuum {
    entry {
        strcpy(CurrentState, ”Low Vacuum”);
        pvPut(CurrentState);
    }

    when (pressure <= .0000049) {
        RoughPump = 0;
        pvPut(RoughPump);
        CryoPump = 1;
        pvPut(CryoPump);
        Valve = 1;
        pvPut(Valve);
    } state high_vacuum

    when (delay(600.0)) {
    } state fault
}
```

Example – State high_vacuum

```
state high_vacuum {
    entry {
        strcpy(CurrentState, "High Vacuum");
        pvPut(CurrentState);
    }

    when (pressure > .0000051) {
        RoughPump = 1;
        pvPut(RoughPump);
        CryoPump = 0;
        pvPut(CryoPump);
        Valve = 0;
        pvPut(Valve);
    } state low_vacuum
}
```

Example – State fault

```
state fault {
    entry {
        strcpy(CurrentState, ”Vacuum Fault”);
        pvPut(CurrentState);
    }
}
```

Building an SNL Program

- Use editor to build the source file. File name must end with “.st” or “.stt”, e.g. “example.st”
- “make” automates these steps:
  - Runs the C preprocessor on “.st” files, but not on “.stt” files.
  - Compiles the state program with SNC to produce C code:
    snc example.st -> example.c
  - Compiles the resulting C code with the C compiler:
    cc example.c -> example.o
  - The object file "example.o” becomes part of the application library, ready to be linked into an IOC binary.
  - The executable file “example” can be created instead.

Running an SNL Program

From an IOC console

- On vxWorks:

```
seq &vacuum_control
```

- On other operating systems:

```
seq vacuum_control
```

- To stop the program:

```
seqStop “vacuum_control”
```

Debugging

Use the sequencer's query commands:

seqShow

- displays information on all running state programs

seqShow `vacuum_control`

- displays detailed information on program

seqChanShow `vacuum_control`

- displays information on all channels

seqChanShow `vacuum_control,”-”`

- displays information on all disconnected channels

Debugging

- Use printf functions to print to the console
```
printf("Here I am in state xyz \n");
```
- Put strings to PVs
```
sprintf(seqMsg1, "Here I am in state xyz");
pvPut(seqMsg1);
```
- On vxWorks/RTEMS you can reload and restart
```
seqStop vacuum_control
```
... edit, recompile ...
```
ld < example.o
seq &vacuum_control
```

Additional Features

- Connection management:

```
when (pvConnectCount() != pvChannelCount())
when (pvConnected(Vin))
```

- Macros:

```
assign Vout to "{unit}:OutputV";
```

  - must use the `+r` compiler option for this if more than one copy of the sequence is running on the same IOC

```
seq &example, "unit=HV01"
```

- Some common SNC compiler options:
  - `+r` make program reentrant (default is `-r`)
  - `-c` don't wait for all channel connections (default is `+c`)
  - `+a` asynchronous **`pvGet()`** (default is `-a`)
  - `-w` don't print compiler warnings (default is `+w`)

Additional Features

- Arbitrary C code can be embedded

  %% escapes one line of C code

  %{

  escape any number of lines of C code

  }%

- Access to channel alarm status and severity:

  pvStatus(*var_name*)
  pvSeverity(*var_name*)

- Queued monitors save CA monitor events in a queue in the order they come in, rather than discarding older values when the program is busy

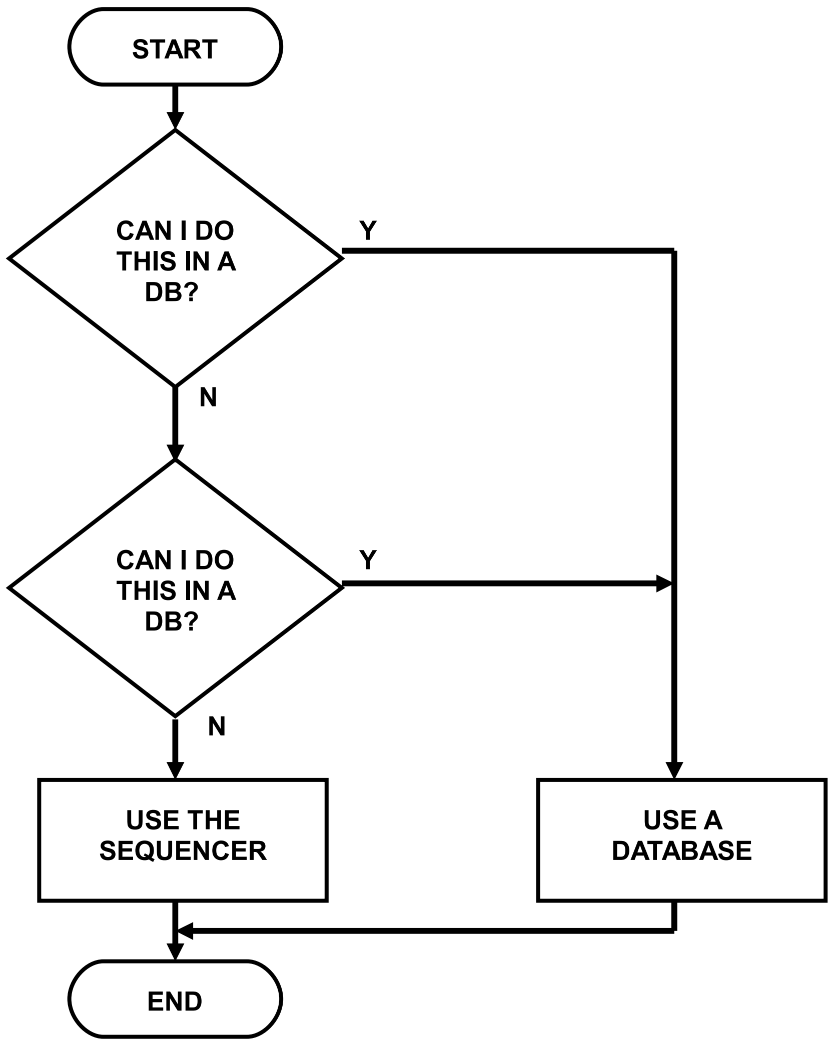
  syncQ *var_name* **to** *event_flag_name [queue_length]*

  pvGetQ(*var_name*)

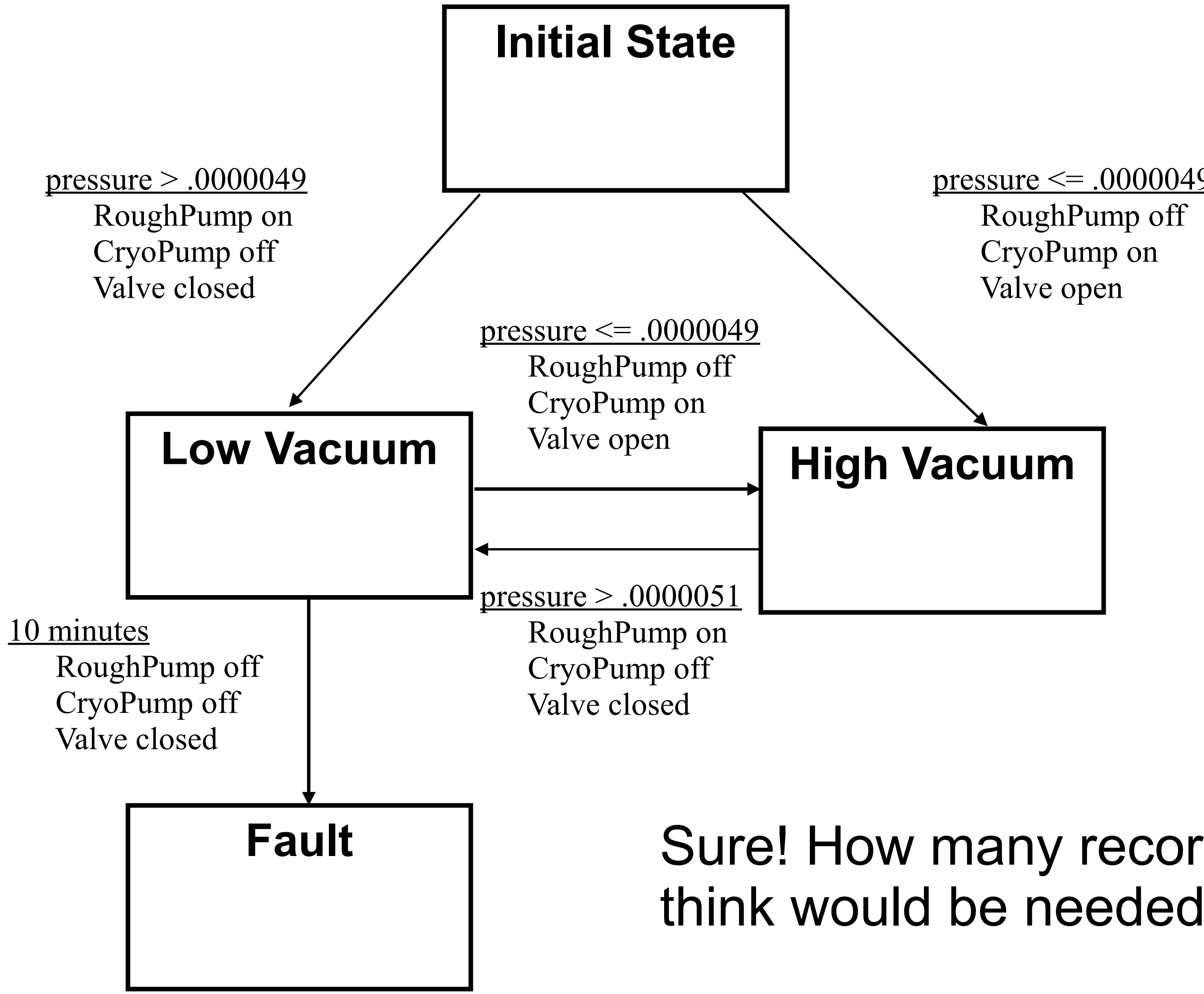
  - removes oldest value from variables monitor queue. Remains true until queue is empty.

  pvFreeQ(*var_name*)

Again: Should I Use the Sequencer?



Can I Do This in a Database?



Sure! How many records do you think would be needed?

Hands-On: SNLexample

wget http://pubweb.bnl.gov/~rlange/SNLexample.tar.gz

This tarball contains:

- These slides (for reference)
- Simulation in vessel.db:
  $(P):pressure, $(P):rough, $(P):valve, $(P):cryo
  Setting $(P):leak to 1 will simulate a leak
- DB state machine in calcfsm.db (for reference)

Create the Example IOC

- Create the Application Development Environment
  mkdir *TOP*; cd *TOP*
  makeBaseApp -t ioc *vacuum* *building environment*
  makeBaseApp -t ioc -i *myIOC* *IOC boot structure*
  cp ../*.db *vacuum*App/Db *copy db files over*
- Add vessel.db to Makefile in *vacuum*App/Db
- make
- Add vessel.db to startup script in iocBoot/ioc*myIOC*/st.cmd (needs a P macro)
- cd iocBoot/ioc*myIOC*; chmod a+x st.cmd
- Run the IOC as ./st.cmd

Next Steps

- Add an edm panel to see stuff working
- Add a db with a single mbbi record ($(P):state) for communicating the current state
- Write the SNL program (in *vacuum*App/src)
- Add the SNL program to Makefile in *vacuum*App/src
- Make (best done on *TOP* level)
- Add starting the state machine to the startup script
- Restart the IOC, debug and have fun

- You can also run vessel.db and calcfsm.db on an IOC to see the db implementation working